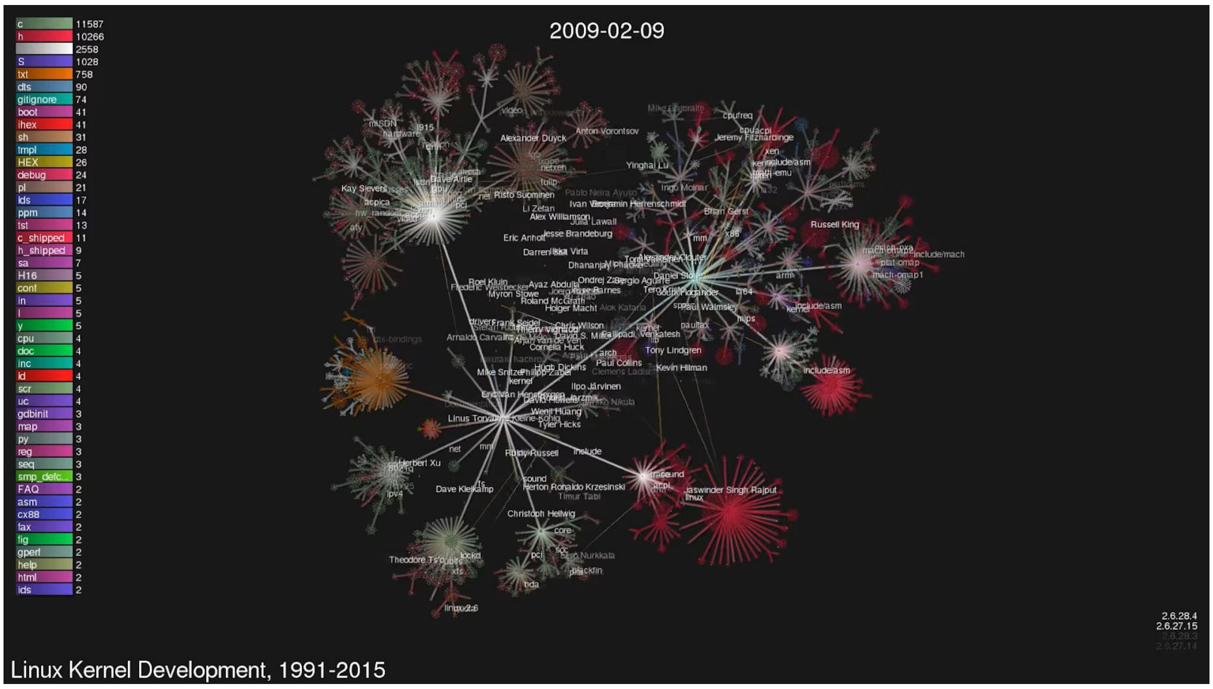
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS33012: Software Engineering
Measuring Software Engineering

John Keaney - 18328855
November 18th - 2020



Source: https://www.youtube.com/watch?v=5iFnzr73XXk
Image Capture of Linux Kernel Development, 1991-2015 - by Darrick Wong

"I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.
I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcdie.libguides.com/plagiarism/ready-steady-write." -John Keaney

Table of Contents

Introduction

"What gets measured, gets managed." - Peter Drucker

Assignment Requirements:

"To deliver a report that considers the **(1) ways in which the software engineering process can be measured** and assessed in terms of measurable data, an **(2) overview of the computational platforms** available to perform this work, the **(3) algorithmic approaches available**, and the **(4) ethics concerns** surrounding this kind of analytics."

Introduction:

Measuring individual and team performance is a strategy as old as time. I remember in Secondary School learning about the world's first factories and management strategies, particularly about Frederick Winslow Taylor, an American Mechanical Engineer who sought to improve industrial efficiency by using stopwatches to gauge employee performance [1]. It only makes sense that in the exponentially growing industry that is the Technology Sector we'd wish to develop the most efficient and effective way of managing our groups to develop the best possible product with the least cost. To create these efficient systems it's in our best interest to discover which employee demonstrates the *"best"* performance. In more traditional professions, such as sales we can determine the best salesperson by volume and gross profit, but for Software Engineering our measurements are not so simple. Not all lines of code are created equally, and it seems that since the dawn of Software Engineering measuring a programmers performance is no easy task.

While there may not be an inherent metric for measuring the performance of a Software Engineer one thing is for certain, Software Engineers provide an abundance of metrics. Although it may not be an immediate indicator of performance, metrics such as number of pull requests by an engineer, customer query satisfaction and rating, total number of total customer issues addressed, lines of code types and many more can be used to help us measure and engineer's ability.

In this report, I hope to show that unfortunately many of the traditional approaches to measuring employee fulfillment do not suit software engineering, why that is, what solutions were proposed and what problems, ethical or not, might arise from measuring engineers a bit too closely.

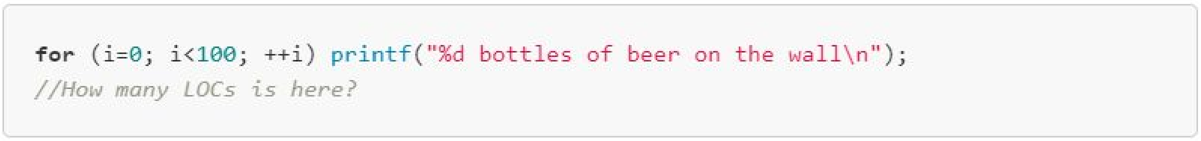
Measurable Data

As mentioned above, there are many different forms of data we can gather from engineers which for the most part can be split into two categories; Code - what are software engineers actually develop, and Social & Environmental Factors - how developers fair in their workplace environment, how well they work with others, capacity for leadership, ability to influence, posture and many more. For now, let's take a look at Code. Later into the report I go into further detail on social.

Lines of Source Code (SLOC) and Other Such Metrics:

With no prior thought on how to measure software engineering, if you were asked how you might measure performance this would most likely be the first thing to come to mind off the top of your head. This metric involved counting the lines of code written by a developer. This is more commonly known as Source Lines of Code, or SLOC. SLOC is usually measured by counting the number of lines of code in a project's codebase and is typically used to predict the amount of effort and time that will be required to develop a program, as well as to estimate the programming productivity once the software is produced [2]. We can separate SLOC into two categories, physical SLOC and logical SLOC. Physical is most likely what you're thinking of, the literal lines of code all counted, even in some instances counting comments and blank lines. Logical SLOC is a little more complicated, it attempts to measure the number of executable expressions (such as operators, functions, etc.). For both SLOC's their specific definitions are tied to specific computer languages.

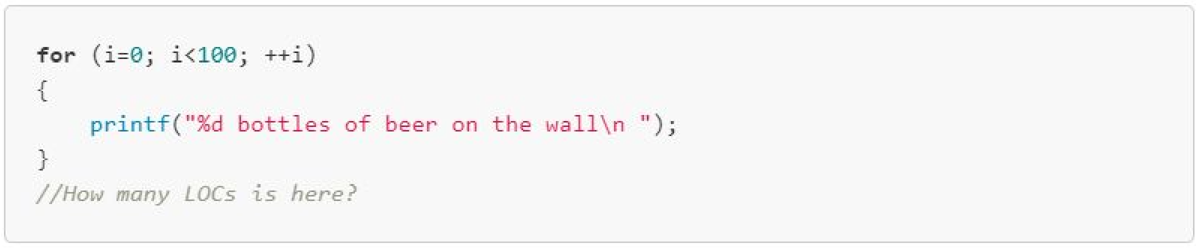
Naturally however, we can see that the more we think about this the less appealing it becomes. Take the code below for example:



```
for (i=0; i<100; ++i) printf("%d bottles of beer on the wall\n");
//How many LOCs is here?
```

Figure 1[4]: https://www.viva64.com/en/t/0086/

This is a simple for loop which sings the campfire song "Bottles of Beer on the Wall", specifying on each line the current number of bottles, incrementing by one. If we take this code at face value, by SLOC analysis we might say that this code featured in Figure 1 consists of two lines of code. Now let's take a look at Figure 2:



```
for (i=0; i<100; ++i)
{
    printf("%d bottles of beer on the wall\n ");
}
//How many LOCs is here?
```

Figure 2[4]: https://www.viva64.com/en/t/0086/

Are Metrics like these even useful?
Notice a difference? Both statements are exactly the same, just of a different form. Immediately we see a way to easily game the system. The number of lines of code is, for the most part, not an accurate measurement of a software engineer's abilities. As Bill Gates once said [3]:

"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."

SLOC also encourages bad practises, such repeating the same functions, or lines of code to meet a specific quota. Let's look at the following example:

```
public static void main(String[] args) {
System.out.print("\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n");
      // Prints some spaces.

   System.out.print("\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n" +
          "\n");
      // Prints some spaces + 1 more.
}
```

These print statements do virtually nothing and are nearly identical to each other, however because the second print statement has one more \n attached to the end it's considered different. With physical SLOC we'd have over 25 lines of code. With logical SLOC we have 2, however we can easily game that also by adding more lessening the number of \n's within the print statement and increasing the number of print statements themselves.

I personally know first hand why SLOC isn't a useful metric. In my first year of college, during our programming project module we made use of a version control tool which maintained a track record of what each individual coded. I personally remember how easy it was to manipulate this tool, by simply uploading a jpeg image to the repository it counted towards an additional 250~ lines or so. Suddenly a well established even distribution of 50/50 could sway towards 30/70 in a matter of minutes.

One final issue with SLOC is how results may vary based on programming languages. Naturally higher level languages usually require fewer lines of code to write virtually the same programs. Such as the inbuilt sorting function for ArrayList, which is only one line, versus the 30 plus lines required to write the same sorting algorithm in Assembly Language.

Similar Metrics:
This metric of measurement is very similar to others so I briefly mention other metrics which fall into the same pitfalls. Some examples include:

- Hours spent programming
- Years learning a language
- Code Coverage and Testing
- Commiting Frequency

All these metrics, fall into the same category of being mostly useless, they have just figures which don't represent the more complicated system and encourage software engineers to write poor quality code and engage in bad practices

On the other hand there are also some similarly simple metrics which provide more value than perhaps the above might:

- Customer Satisfaction / Rating
- Bug Catching / Solving
- Performance Over Time / Ability to adapt

What makes these metrics superior to the ones above is that they are much more difficult to cheat on. They are presented as unique scenarios. A programmer can only solve so many similar bugs until he is required to learn something new and use his skills. Performance over time is also difficult to cheat on simply because it requires a developer to trick their managers and reviewers for a long period of time consecutively, which in most scenarios they are caught quick enough. Still the above metrics still have pitfalls, customer satisfaction for example can vary greatly with some engineers receiving a wave of easy issues to solve and other engineers receiving a wave of hard issues. It wouldn't be right to judge these two engineers against each other, because even if they have the same skill level and set, their end results would differ.

It would seem that modern problems require modern solutions. So let's take a look at some.

Agile Process Metrics:

As time has gone on in my degree, I've noticed more and more how important the Agile and SCRUM Master development process is. It seems to be the most popular development strategy deployed especially by the small teams I've worked with. Unlike the metrics mentioned previously, which are mostly software development based, Agile favours delivered business value [5]. For the purpose of this report I'll discuss what I believe to be the two most important factors; Velocity/Cycle/Lean Time and Sprint/Release Burn-Down Charts.

Velocity/Cycle/Lean Time:

Velocity/Cycle/Lean Time all refer to the time it takes to complete a project, each term referring to different segments of the cycle. Here's a graphic to explain them:

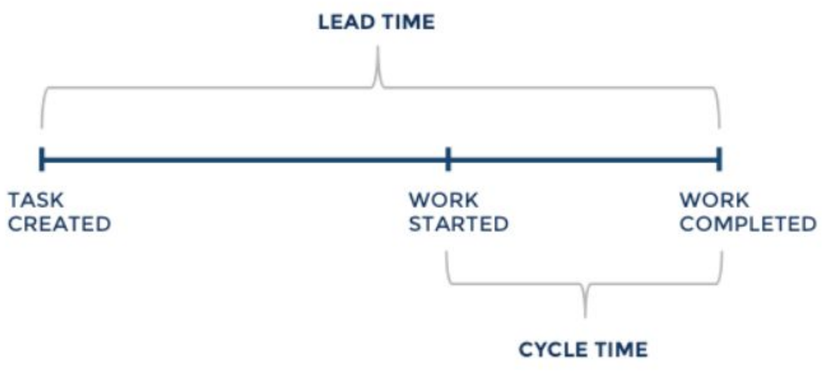


Figure 3[6]: https://screenful.com/blog/software-development-metrics-cycle-time

To look specifically at Cycle Time, Cycle time is a measure of the elapsed time when work starts on an item (story, task, bug etc.) until it's ready for delivery. Cycle time tells how long (in calendar time) it takes to complete a task[6].

Why are these Metrics useful? :

This is specifically very useful for clients. Instead of engineers quoting a guess, groups can look at past projects to see how long it takes to complete previous work and provide a somewhat accurate estimate. An indication of an engineer's performance by supplying them with tasks of similar lengths help to avoid unwanted surprises in the future such as bottlenecks. Not only that but, by looking at cycle times of previous projects, managers and SCRUM masters are able to tell what areas of the workflow are particularly lacking and continue to make improvements to fasten the cycle. With Agile development this is particularly important considering how detrimental feedback for clients is.

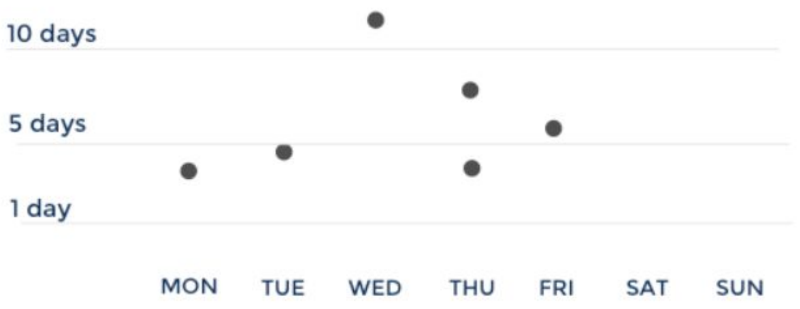


Figure 4[6]: https://screenful.com/blog/software-development-metrics-cycle-time

In the graph above (Figure 4) we see specific tasks being completed for each day of the week. By using the cycle time metric and a scatter plot like the one above it becomes much easier to point out and isolate specific weaknesses in development. Take a look at Wednesday for example. A manager would be able to tell that Wednesday tasks take the most amount of time so how might we adjust these tasks to create less bottleneck and allow for smoother flow.

Now, by improving and shortening cycle time, developers can introduce new features and improvements at a greater rate which for many is their bottom line for a client. By making use of Little's Law [7] we know that when keeping the same pace of work, if current work in progress goes down then the cycle time of a feature goes down as well. Formally:

$$Speed\ of\ Product\ Iterations\ = Average\ Cycle\ Time\ = \frac{Average\ Work\ in\ Progress}{Average\ Throughput}$$

Sprint/Release Burn-Down Charts:

A burndown chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.[8]

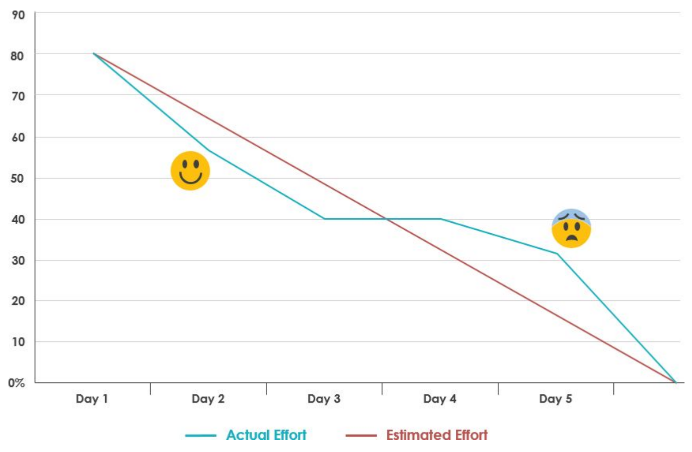


Figure 5: https://www.visual-paradigm.com/scrum/scrum-burndown-chart/

How can we use these charts to keep on schedule?
As the name implies Sprint/Release Burn-Down Charts make the work of a group visible. These charts are a "must" for SCRUM teams. It's a representation of the rate at which work is completed and work is left to do. It works as an effective tool that can be used to show Team progress towards a Sprint Goal, showing how much work remains rather than time spent. If the burndown line at mid-Sprint point for the team is not moving downwards then the team needs to act quickly by implementing an Emergency Procedure pattern.

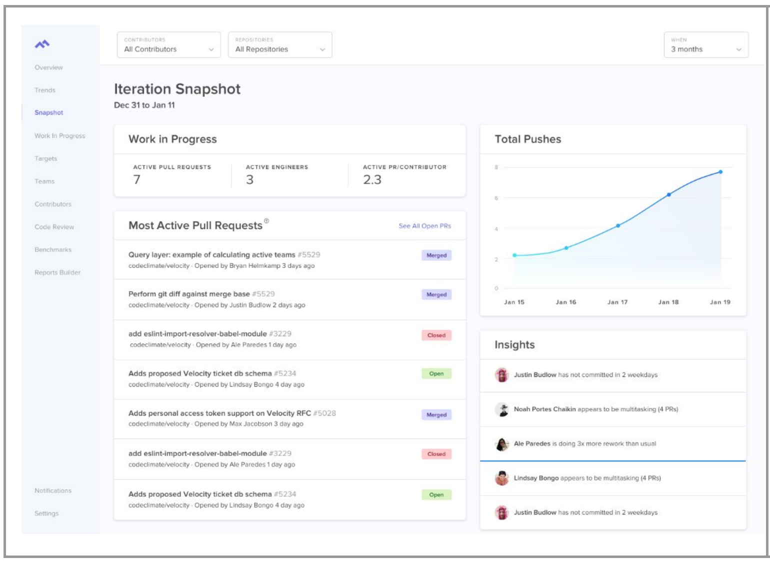
Ultimately these charts are so quintessential for the following reasons:
- Monitoring the project scope creep
- Keeping the team running on schedule
- Comparing the planned work against the team progression

Algorithmic Approaches to Measuring SW, Platforms and Frameworks Available & Their Ethics

Unfortunately, I'm not too familiar with algorithmic approaches to measuring software engineering as the information behind this is a science in it's own right. I'll be keeping this section short and to the point on what platforms I've looked into and how they assess software development.

Platform #1: Code Climate & Code Climate's Rating System.

Code Climate is a company founded in 2011 with the goal of solving an issue common among software engineering leaders, how is it possible to maintain quality of code as the project progresses. To solve this issues they created two products; Quality and Velocity.[9]



Iteration Snapshot
Dec 31 to Jan 11

Work in Progress

ACTIVE PULL REQUESTS	ACTIVE ENGINEERS	ACTIVE PR/CONTRIBUTOR
7	3	2.3

Most Active Pull Requests® — See All Open PRs

Query layer: example of calculating active teams #5529 — Merged
codeclimate/velocity · Opened by Bryan Helmkamp 3 days ago

Perform git diff against merge base #5529 — Merged
codeclimate/velocity · Opened by Justin Budlow 2 days ago

add eslint-import-resolver-babel-module #3229 — Closed
codeclimate/velocity · Opened by Ale Paredes 1 day ago

Adds proposed Velocity ticket db schema #5234 — Open
codeclimate/velocity · Opened by Lindsay Bongo 4 day ago

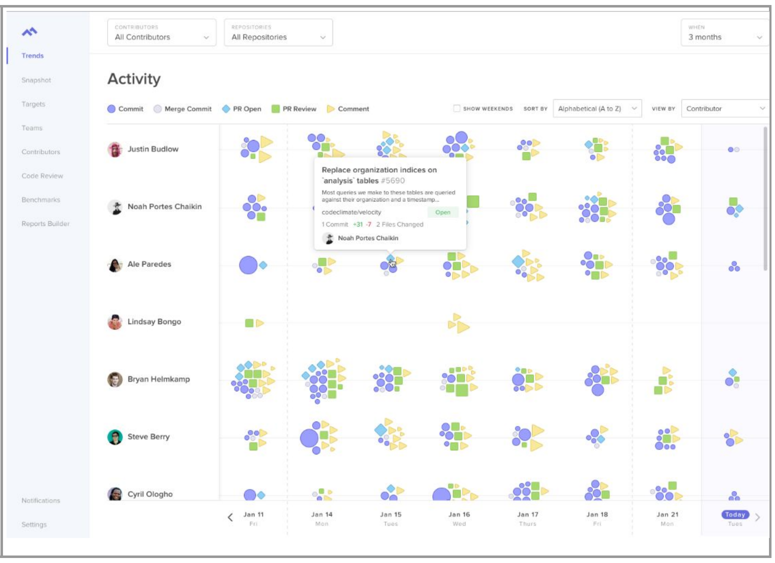
Adds personal access token support on Velocity RFC #5028 — Merged
codeclimate/velocity · Opened by Max Jacobson 3 day ago

add eslint-import-resolver-babel-module #3229 — Closed
codeclimate/velocity · Opened by Ale Paredes 1 day ago

Adds proposed Velocity ticket db schema #5234 — Open
codeclimate/velocity · Opened by Lindsay Bongo 4 day ago

Insights

Justin Budlow has not committed in 2 weekdays
Noah Portes Chaikin appears to be multitasking (4 PRs)
Ale Paredes is doing 3x more rework than usual
Lindsay Bongo appears to be multitasking (4 PRs)
Justin Budlow has not committed in 2 weekdays



Replace organization indices on `analysis` tables #5690
Most queries we make to these tables are queried against their organization and a timestamp...
codeclimate/velocity — Open
1 Commit +31 -7 2 Files Changed
Noah Portes Chaikin

Figure 6: Code Climate UI: https://codeclimate.com/customers/tangoe/

Quality is an automated code review tool, which helps developers save time and improve the maintainability of their codebases. Quality helps by automatically reviewing commits, checking test coverage, track code coverage and identifying hot spots in the code where problems may occur.[10]

Velocity, analyzes all the data from your GitHub repos and provides you with heads-up displays, real-time analytics, and custom reports to give you a clearer perspective on how your engineering team is working.[11]

Code Climate 10-Point Technical Debt Assessment & Algorithm:

Code Climate uses a 10-Point Technical Debt Assessment to provide a clear cut user friendly ranking system. The ranking system displays focus on two priorities for code: maintainability and test coverage:

- Test coverage: Take the covered lines of code compared to the total "coverable" lines of code as a percentage, and map it to a letter grade from A to F.
- Technical debt: Static analysis can examine a codebase for potential structural issues, but different tools tend to focus on different (and often overlapping) problems. There has never been a single standard, and so Code Climate set out to create one.

Code Climate uses the following conditions to determine the code's rating:

1. Argument count – Methods or functions defined with a high number of arguments
2. Complex boolean logic – Boolean logic that may be hard to understand
3. File length – Excessive lines of code within a single file
4. Identical blocks of code – Duplicate code which is syntactically identical (but may be formatted differently)
5. Method count – Classes defined with a high number of functions or methods
6. Method length – Excessive lines of code within a single function or method
7. Nested control flow – Deeply nested control structures like if or case
8. Return statements – Functions or methods with a high number of return statements
9. Similar blocks of code – Duplicate code which is not identical but shares the same structure (e.g. variable names may differ)
10. Method complexity – Functions or methods that may be hard to understand

[12]

In general Code Climate makes use of a number of tools and algorithms to eliminate potential future problems.

Ethical Concerns:

There is not much to discuss about Code Climate's ethical policies as I believe most would consider them to be only ethical. Code Climate merely extracts data that is already available to managers and SCRUM masters and makes it more understandable and readable. The data it collects is already privately owned by the group rather than the developers. A large proportion of the code behind Code Climates programs is even open source, available to others on github: https://github.com/codeclimate/codeclimate[13] so individuals are usually able to fairly decide whether or not how their data is processed is fair. Many, including myself might believe that Code Climate encourages greater ethics in the workplace. It allows users to see what areas of their workflow are lacking and presents clear results, meaning that unwanted human factors such as personal prejudice have little chance of getting in the way. Code Climate allows developers to reach their full potential rather than be held back by issues they are unable to see.

Platform #2: Fitbit and Corporate Wellness

The next platform I wish to discuss is Fitbit. With Code Climate I focused on how I feel Software Engineering should be measured and made useful while remaining ethical. Fitbit provides useful information for SCRUM Masters and managers however as a result it sacrifices ethics in ways that most would deem intrusive.

A Fitbit is an activity tracker, usually worn on the wrist, which can track the distance you walk, run, swim or cycle, as well as the number of calories you burn and take in. Some also monitor your heart rate and sleep quality. Fitbit help their users to keep track of [14]:

1. Heart Rate
2. Intensive Exercise
3. Low effort Exercise
4. Sleep and Sleep Patterns
5. Caloric intake and expenditure

Newer models of fitbit also come with:

6. Voice Recognition
7. Online Search Engine / Browser History
8. Location and GPS history
9. And in general all the capabilities of a modern smart watch.

Health Tracking:

Companies making use of Fitbits include both technology industry and non-technical industry firms, including but not limited to [15]; BP Oil, Bangor Savings Bank, Cisco, Indiana University and many more. Most would agree that data such as this is nothing less than extreme and creeps too far into personal areas of people's life. The first 5 factors go into great detail into an individual's health. Fitbit claims that this data can be used to help prevent insurance claims and cut health insurance costs however a company that has nothing to do with health should not be collecting that data from it's users, it simply does not belong to them.

Tracking too much:

What's worse is the other data collected, i.e points six to nine listed above. This data is quite useful to companies as with it they are able to collect numerous data points on an individual and as a result, create a surprising accurate estimate of what the individual is like. But as a result the person wearing the fitbit has virtually no privacy whatsoever. The browser history and location component is particularly despicable. With this, employers are able to obtain information on the employee that would otherwise remain secret. With information like this, and the hierarchical structure of most workplaces, a worker has almost no control in their profession and will need to suit to the whim of their employers.

Your Private Information, Open to the World with Fitbit:

Finally and most importantly, as you might have guessed this data is not safe. This data is used by both employers and Fitbit but as recently as February 2020 Fitbit underwent a data breach and the data of almost two million of its users was exposed to cyber criminals and as such sold on the dark web [16]. This isn't the first instance of data leaks from Fitbits [17] and it most likely will not be the last.

Conclusion and Personal Experience on Measuring Workplace Ethics

Conclusion:

This report began with a discussion on traditional metrics of measuring software engineering and how most of them simply do not scale to what is needed in accurate measurement. Then we looked at the solution to this issue and why an Agile methodology may be useful not only in developing software quickly but also how we might decrease overall cycle time altogether. I've found that measuring Agile is a particularly useful way of measuring software development.

The next section of this report revolved around the platform of Code Climate and how their framework is not only useful for monitoring and measuring software teams progress but also the quality of software and code written. Most would consider their approach to be ethical for both workers and the world in general.

Finally the report covered Fitbit and their Corporate Wellness program. I concluded that this program was invasive and unnecessary for an individual's professional life. Also my research drew the fact that such measures are unsafe and put the workers' privacy at risk.

Personal Experience:

For a brief period of time in the past I worked as a packager at a certain packaging warehouse. Our supervisors made us aware that the number of packages we sorted would be counted by a scanner badge and would affect our reviews at the end of each month. Positive performance would mean a workplace promotion while poor performance would result in a poor review and even potentially a termination. What was worse about these scanners is that they also took track of time so it was rather easy to tell when an individual took bathroom breaks and the length of these breaks. Luckily our supervisors were considerate but I have heard of accounts of individuals at other warehouses who have received warnings because of the length of their breaks, sometimes even only because a few minutes were considered excessive. I understand from a corporate perspective that time is money however most would agree that this is an invasion of privacy. From my research for this paper I've gained an appreciation for valid measurement of software engineering and that a company's maximum potential doesn't necessarily need to require invasive procedures.

Bibliography

[1] Frederick Winslow Taylor: Stopwatches and Employee Performance
https://hbr.org/1988/11/the-same-old-principles-in-the-new-manufacturing
[2] Measuring Source Lines of Code: https://www.viva64.com/en/t/0086/
[3] Bill Gates Quote:
https://www.goodreads.com/quotes/536587-measuring-programming-progress-by-lines-of-code-is-like-measuring#:~:text=%E2%80%9CMeasuring%20programming%20progress%20by%20lines%20of%20code%20is%20like,aircraft%20building%20progress%20by%20weight.%E2%80%9D
[4] SLOC Examples: https://www.viva64.com/en/t/0086/
[5] Agile Development Metrics:
https://insights.sei.cmu.edu/sei_blog/2014/09/agile-metrics-seven-categories.html
[6] Agile Cycle Image: https://screenful.com/blog/software-development-metrics-cycle-time
[7] Little's Law: https://en.wikipedia.org/wiki/Little%27s_law
[8] Sprint Chart: https://www.visual-paradigm.com/scrum/scrum-burndown-chart/
[9] Code Climate: About Us: https://codeclimate.com/about/
[10] Code Climate: Quality: https://codeclimate.com/quality/
[11] Code Climate: Velocity: https://codeclimate.com/
[12] 10-Point Ranking System: https://codeclimate.com/blog/10-point-technical-debt-assessment/
[13] Code Climate Open Source Code: https://github.com/codeclimate/codeclimate
[14] Fitbit for Corporate Use:
https://www.fitbit.com/content/assets/group-health/FitbitWellness_InfoSheet.pdf
[15] Companies using Fitbit:
https://www.featuredcustomers.com/vendor/fitbit-health-solutions/customers
[16] Fitbit Data Breach:
https://hackernoon.com/2-million-fitbit-accounts-was-exposed-by-cybercriminals-aa7u36pj
[17] Previous Fitbit Data Leak:
https://hackernoon.com/2-million-fitbit-accounts-was-exposed-by-cybercriminals-aa7u36pj